Защита лабораторной работы №2. Структуры данных.

Ишанова А.И.

19 ноября 2022

RUDN University, Moscow, Russian Federation

Прагматика выполнения лабораторной работы

Прагматика выполнения лабораторной работы

- изучение некоторых структур данных в Julia
  - кортеж
  - словарь
  - множество
  - массив
- приобретения навыков работы с этими структурами данных

Цель выполнения лабораторной работы

Цель выполнения лабораторной работы

Изучить несколько структур данных, реализованных в Julia, научиться применять их и операции над ними для решения задач.

Выполнение лабораторной работы

Повторение примеров

Задание 1. Повторение примеров из раздела 2.2.

1. Повторяем примеры с кортежами. (fig. 1)

```
[1]: # пустой кортеж:
     ()

[1]: ()

[2]: # кортеж из элементов типа String:
     favoritelang = ("Python","Julia","R")

[2]: ("Python", "Julia", "R")

[3]: # кортеж из целых чисел:
     x1 = (1, 2, 3)

[3]: (1, 2, 3)

[5]: # кортеж из элементов разных типов:
     x2 = (1, 2.0, "tmp")

[5]: (1, 2.0, "tmp")

[6]: # именованный кортеж:
     x3 = (a=2, b=1+2)

[6]: (a = 2, b = 3)

[7]: # длина кортежа x2:
     length(x2)

[7]: 3

[8]: # обратиться к элементам кортежа x2:
     x2[1], x2[2], x2[3]

[8]: (1, 2.0, "tmp")

[9]: # произвести какую-либо операцию (сложение)
     # с вторым и третьим элементами кортежа x1:
     c = x1[2] + x1[3]

[9]: 5

[10]: # обращение к элементам именованного кортежа x3:
      x3.a, x3.b, x3[2]

[10]: (2, 3, 3)

[11]: # проверка вхождения элементов tmp и 0 в кортеж x2
      # (два способа обращения к методу in()):
      in("tmp", x2), 0 in x2

[11]: (true, false)
```

Figure 1: Примеры с кортежами

Повторение примеров

2. Потворяем примеры со словарями. (fig. 2)

```
[23]: # создать словарь с именем phonebook:
phonebook = Dict("Иванов И.И." => ("867-5309","333-5544"),
    "Бухгалтерия" => "555-2368")

[23]: Dict{String, Any} with 2 entries:
  "Бухгалтерия" => "555-2368"
  "Иванов И.И." => ("867-5309", "333-5544")

[13]: # вывести ключи словаря:
keys(phonebook)

[13]: KeySet for a Dict{String, Any} with 2 entries. Keys:
  "Бухгалтерия"
  "Иванов И.И."

[14]: # вывести значения элементов словаря:
values(phonebook)

[14]: ValueIterator for a Dict{String, Any} with 2 entries. Values:
  "555-2368"
  ("867-5309", "333-5544")

[15]: # вывести заданные в словаре пары "ключ - значение":
pairs(phonebook)

[15]: Dict{String, Any} with 2 entries:
  "Бухгалтерия" => "555-2368"
  "Иванов И.И." => ("867-5309", "333-5544")

[16]: # проверка вхождения ключа в словарь:
haskey(phonebook, "Иванов И.И.")

[16]: true

[24]: # добавить элемент в словарь:
phonebook["Сидоров П.С."] = "555-3344"

[24]: "555-3344"

[25]: # удалить ключ и связанные с ним значения из словаря
pop!(phonebook, "Иванов И.И.")

[25]: ("867-5309", "333-5544")

[26]: # Объединение словарей (функция merge()):
a = Dict("foo" => 0.0, "bar" => 42.0);
b = Dict("baz" => 17, "bar" => 13.0);
merge(a, b), merge(b,a)

[26]: (Dict{String, Real}("bar" => 13.0, "baz" => 17, "foo" => 0.0), Dict{String, R
eal}("bar" => 42.0, "baz" => 17, "foo" => 0.0))
```

Figure 2: Примеры со словарями

Повторение примеров

3. Повторяем примеры с множествами. (fig. 3 - fig. 4)

```
[27]: # создать множество из четырёх целочисленных значений:
      A = Set([1, 3, 4, 5])

[27]: Set{Int64} with 4 elements:
        5
        4
        3
        1

[28]: # создать множество из 11 символьных значений:
      B = Set("abrakadabra")

[28]: Set{Char} with 5 elements:
        'a'
        'd'
        'r'
        'k'
        'b'

[29]: # проверка эквивалентности двух множеств:
      S1 = Set([1,2]);
      S2 = Set([3,4]);
      issetequal(S1,S2)

[29]: false

[30]: S3 = Set([1,2,2,3,1,2,3,2,1]);
      S4 = Set([2,3,1]);
      issetequal(S3,S4)

[30]: true

[31]: # объединение множеств:
      C=union(S1,S2)

[31]: Set{Int64} with 4 elements:
        4
        2
        3
        1

[32]: # пересечение множеств:
      D = intersect(S1,S3)

[32]: Set{Int64} with 2 elements:
        2
        1

[33]: # разность множеств:
      E = setdiff(S3,S1)

[33]: Set{Int64} with 1 element:
        3
```

Figure 3: Примеры с множествами

Повторение примеров

```
[34]: # проверка вхождения элементов одного множества в другое:
      issubset(S1,S4)

[34]: true

[35]: # добавление элемента в множество:
      push!(S4, 99)

[35]: Set{Int64} with 4 elements:
        2
        99
        3
        1

[36]: # удаление последнего элемента множества:
      pop!(S4)

[36]: 2
```

Figure 4: Примеры с множествами (2 часть)

Повторение примеров

4. Повторяем примеры работы с массивами. (fig. 5 - fig. 11)

```
[37]: # создание пустого массива с абстрактным типом:
      empty_array_1 = []

[37]: Any[]

[39]: # создание пустого массива с конкретным типом:
      empty_array_2 = (Int64)[]

[39]: Int64[]

[40]: empty_array_3 = (Float64)[]

[40]: Float64[]

[41]: # вектор-столбец:
      a = [1, 2, 3]

[41]: 3-element Vector{Int64}:
       1
       2
       3

[42]: # вектор-строка:
      b = [1 2 3]

[42]: 1×3 Matrix{Int64}:
       1  2  3

[43]: # многомерные массивы (матрицы):
      A = [[1, 2, 3] [4, 5, 6] [7, 8, 9]]

[43]: 3×3 Matrix{Int64}:
       1  4  7
       2  5  8
       3  6  9

[44]: B = [[1 2 3]; [4 5 6]; [7 8 9]]

[44]: 3×3 Matrix{Int64}:
       1  2  3
       4  5  6
       7  8  9

[45]: # одномерный массив из 8 элементов (массив $1 \times 8$)
      # со значениями, случайно распределёнными на интервале [0, 1):
      c = rand(1,8)

[45]: 1×8 Matrix{Float64}:
       0.209468  0.4564  0.716093  0.980613  …  0.0162479  0.548266  0.456862
```

Figure 5: Примеры работы с массивами

Повторение примеров

```
[46]: # многомерный массив $2 \times 3$ (2 строки, 3 столбца) элементов
      # со значениями, случайно распределёнными на интервале [0, 1):
      C = rand(2,3);

[47]: C

[47]: 2×3 Matrix{Float64}:
       0.113001   0.941538  0.804419
       0.0108865  0.935917  0.653456

[48]: # трёхмерный массив:
      D = rand(4, 3, 2)

[48]: 4×3×2 Array{Float64, 3}:
      [:, :, 1] =
       0.324993  0.920096   0.0513952
       0.328887  0.0933765  0.0491648
       0.879014  0.2423     0.362187
       0.583426  0.360941   0.213788

      [:, :, 2] =
       0.483531  0.347641  0.669412
       0.095872  0.656512  0.402238
       0.609596  0.412485  0.296019
       0.133062  0.652427  0.532998

[49]: # массив из квадратных корней всех целых чисел от 1 до 10:
      roots = [sqrt(i) for i in 1:10]

[49]: 10-element Vector{Float64}:
       1.0
       1.4142135623730951
       1.7320508075688772
       2.0
       2.23606797749979
       2.449489742783178
       2.6457513110645907
       2.8284271247461903
       3.0
       3.1622776601683795

[50]: # массив с элементами вида 3*x^2,
      # где x – нечётное число от 1 до 9 (включительно)
      ar_1 = [3*i^2 for i in 1:2:9]

[50]: 5-element Vector{Int64}:
          3
         27
         75
        147
        243
```

Figure 6: Примеры работы с массивами (часть 2)

Повторение примеров

```
[51]: # массив квадратов элементов, если квадрат не делится на 5 или 4:
ar_2=[i^2 for i=1:10 if (i^2%5!=0 && i^2%4!=0)]

[51]: 4-element Vector{Int64}:
  1
  9
 49
 81

[52]: # одномерный массив из пяти единиц:
ones(5)

[52]: 5-element Vector{Float64}:
 1.0
 1.0
 1.0
 1.0
 1.0

[53]: # двумерный массив 2x3 из единиц:
ones(2,3)

[53]: 2×3 Matrix{Float64}:
 1.0  1.0  1.0
 1.0  1.0  1.0

[54]: # одномерный массив из 4 нулей:
zeros(4)

[54]: 4-element Vector{Float64}:
 0.0
 0.0
 0.0
 0.0

[55]: # заполнить массив 3x2 цифрами 3.5
fill(3.5,(3,2))

[55]: 3×2 Matrix{Float64}:
 3.5  3.5
 3.5  3.5
 3.5  3.5

[56]: # заполнение массива посредством функции repeat():
repeat([1,2],3,3)
repeat([1 2],3,3)

[56]: 3×6 Matrix{Int64}:
 1  2  1  2  1  2
 1  2  1  2  1  2
 1  2  1  2  1  2
```

Figure 7: Примеры работы с массивами (часть 3)

Повторение примеров

```
[57]: # преобразование одномерного массива из целых чисел от 1 до 12
      # в двумерный массив  2x6
      a = collect(1:12)
      b = reshape(a,(2,6))
```

```
[57]: 2×6 Matrix{Int64}:
       1  3  5  7   9  11
       2  4  6  8  10  12
```

```
[58]: # транспонирование
      b'
```

```
[58]: 6×2 adjoint(::Matrix{Int64}) with eltype Int64:
        1   2
        3   4
        5   6
        7   8
        9  10
       11  12
```

```
[59]: # транспонирование
      c = transpose(b)
```

```
[59]: 6×2 transpose(::Matrix{Int64}) with eltype Int64:
        1   2
        3   4
        5   6
        7   8
        9  10
       11  12
```

```
[60]: #  массив 10x5 целых чисел в диапазоне [10, 20]:
      ar = rand(10:20, 10, 5)
```

```
[60]: 10×5 Matrix{Int64}:
       14  14  16  11  17
       20  11  17  15  16
       18  11  13  19  11
       19  13  13  18  10
       17  17  16  19  18
       14  18  19  14  15
       20  19  12  14  16
       12  17  11  13  15
       11  15  15  15  12
       16  10  11  15  20
```

Figure 8: Примеры работы с массивами (часть 4)

Повторение примеров

```
[61]: # выбор всех значений строки в столбце 2:
ar[:, 2]
```

```
[61]: 10-element Vector{Int64}:
 14
 11
 11
 13
 17
 18
 19
 17
 15
 10
```

```
[62]: # выбор всех значений в столбцах 2 и 5:
ar[:, [2, 5]]
```

```
[62]: 10×2 Matrix{Int64}:
 14  17
 11  16
 11  11
 13  10
 17  18
 18  15
 19  16
 17  15
 15  12
 10  20
```

```
[63]: # все значения строк в столбцах 2, 3 и 4:
ar[:, 2:4]
```

```
[63]: 10×3 Matrix{Int64}:
 14  16  11
 11  17  15
 11  13  19
 13  13  18
 17  16  19
 18  19  14
 19  12  14
 17  11  13
 15  15  15
 10  11  15
```

```
[64]: # значения в строках 2, 4, 6 и в столбцах 1 и 5:
ar[[2, 4, 6], [1, 5]]
```

```
[64]: 3×2 Matrix{Int64}:
 20  16
 19  10
 14  15
```

Figure 9: Примеры работы с массивами (часть 5)

Повторение примеров

```
[65]: # значения в строке 1 от столбца 3 до последнего столбца:
ar[1, 3:end]

[65]: 3-element Vector{Int64}:
 16
 11
 17

[66]: # сортировка по столбцам:
sort(ar,dims=1)

[66]: 10×5 Matrix{Int64}:
 11  10  11  11  10
 12  11  11  13  11
 14  11  12  14  12
 14  13  13  14  15
 16  14  13  15  15
 17  15  15  15  16
 18  17  16  15  16
 19  17  16  18  17
 20  18  17  19  18
 20  19  19  19  20

[67]: # сортировка по строкам:
sort(ar,dims=2)

[67]: 10×5 Matrix{Int64}:
 11  14  14  16  17
 11  15  16  17  20
 11  11  13  18  19
 10  13  13  18  19
 16  17  17  18  19
 14  14  15  18  19
 12  14  16  19  20
 11  12  13  15  17
 11  12  15  15  15
 10  11  15  16  20

[68]: # поэлементное сравнение с числом
# (результат – массив логических значений):
ar .> 14

[68]: 10×5 BitMatrix:
 0  0  1  0  1
 1  0  1  1  1
 1  0  0  1  0
 1  0  0  1  0
 1  1  1  1  1
 0  1  1  0  1
 1  1  0  0  1
 0  1  0  0  1
 0  1  1  1  0
 1  0  0  1  1
```

Figure 10: Примеры работы с массивами (часть 6)

Повторение примеров

```
[69]: # возврат индексов элементов массива, удовлетворяющих условию:
      findall(ar .> 14)
```

```
[69]: 29-element Vector{CartesianIndex{2}}:
       CartesianIndex(2, 1)
       CartesianIndex(3, 1)
       CartesianIndex(4, 1)
       CartesianIndex(5, 1)
       CartesianIndex(7, 1)
       CartesianIndex(10, 1)
       CartesianIndex(5, 2)
       CartesianIndex(6, 2)
       CartesianIndex(7, 2)
       CartesianIndex(8, 2)
       CartesianIndex(9, 2)
       CartesianIndex(1, 3)
       CartesianIndex(2, 3)
       ⋮
       CartesianIndex(3, 4)
       CartesianIndex(4, 4)
       CartesianIndex(5, 4)
       CartesianIndex(9, 4)
       CartesianIndex(10, 4)
       CartesianIndex(1, 5)
       CartesianIndex(2, 5)
       CartesianIndex(5, 5)
       CartesianIndex(6, 5)
       CartesianIndex(7, 5)
       CartesianIndex(8, 5)
       CartesianIndex(10, 5)
```

Figure 11: Примеры работы с массивами (часть 7)

Выполнения задания для самостоятельной работы - 1

1. Даны множества: A = {0,3,4,9}, B = {1,3,4,7}, C = {0,1,2,4,7,8,9}. Нашли $P = A \cap B \cup A \cap B \cup A \cap C \cup B \cap C$. (fig. 12)

```
[ ]: # ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

[71]: #1.
      A = Set([0, 3, 4, 9])
      B = Set([1, 3, 4, 7])
      C = Set([0, 1, 2, 4, 7, 8, 9])

      AandB = intersect(A,B)
      AandC = intersect(A,C)
      BandC = intersect(B,C)

      p1 = union(AandB, AandB)
      p2 = union(p1, AandC)

      P = union(p2, BandC)

[71]: Set{Int64} with 6 elements:
        0
        4
        7
        9
        3
        1
```

Figure 12: Поиск множества P

Выполнения задания для самостоятельной работы - 2

2. Привели свои примеры с выполнением операций над множествами элементов разных типов. (fig. 13 - fig. 14)

```
[72]: #2.
      Q = Set([1, 2, 3, 4])
      W = Set([1.0, 3.3, 7.6])
      E = Set("qwerty")

[72]: Set{Char} with 6 elements:
        'w'
        'y'
        'e'
        'r'
        't'
        'q'

[73]: issetequal(Q,W)

[73]: false

[74]: union(Q,W,E)

[74]: Set{Any} with 12 elements:
        'w'
        7.6
        'e'
        1.0
        4
        2
        'y'
        3.3
        'r'
        't'
        3
        'q'

[75]: intersect(Q,W)

[75]: Set{Float64} with 1 element:
        1.0

[76]: setdiff(Q,W)

[76]: Set{Int64} with 3 elements:
        4
        2
        3

[77]: issubset(Q,W)

[77]: false
```

Figure 13: Примеры с выполнением операций над множествами элементов разных типов

Выполнения задания для самостоятельной работы - 2

```
[78]: push!(E, 'u')

[78]: Set{Char} with 7 elements:
        'w'
        'y'
        'u'
        'e'
        'r'
        't'
        'q'

[79]: pop!(E)

[79]: 'w': ASCII/Unicode U+0077 (category Ll: Letter, lowercase)
```

Figure 14: Примеры с выполнением операций над множествами элементов разных типов (часть 2)

Выполнения задания для самостоятельной работы - 3 (3.1)

3. Создали разными способами:

   3.1. массив(1,2,3,...,N−1,N), с N = 21; (fig. 15)

```
[80]: #3.
      #1) N=21
      [i for i in 1:21]

[80]: 21-element Vector{Int64}:
       1
       2
       3
       4
       5
       6
       7
       8
       9
      10
      11
      12
      13
      14
      15
      16
      17
      18
      19
      20
      21

[81]: collect(1:21)

[81]: 21-element Vector{Int64}:
       1
       2
       3
       4
       5
       6
       7
       8
       9
      10
      11
      12
      13
      14
      15
      16
      17
      18
      19
      20
      21
```

Figure 15: Создание массива от 1 до 21 2-мя способами

Выполнения задания для самостоятельной работы - 3.2

Задание 2. Задания для самостоятельной работы.

3.2. массив(N,N−1...,2,1), с N = 21; (fig. 16)

```
[83]: #2)
      [i for i in 21:-1:1]

[83]: 21-element Vector{Int64}:
       21
       20
       19
       18
       17
       16
       15
       14
       13
       12
       11
       10
        9
        8
        7
        6
        5
        4
        3
        2
        1

[85]: sort(collect(1:21), rev=true)

[85]: 21-element Vector{Int64}:
       21
       20
       19
       18
       17
       16
       15
       14
       13
       12
       11
       10
        9
        8
        7
        6
        5
        4
        3
        2
        1
```

Figure 16: Создание массива от 21 до 1 2-мя способами

Выполнения задания для самостоятельной работы - 3.3

3.3. массив(1,2,3,...,N−1,N,N−1,...,2,1), с N = 21;(fig. 17-fig. 18)

```
[163]: #3)
       append!(collect(1:21), sort(collect(1:20), rev = true))

[163]: 41-element Vector{Int64}:
        1
        2
        3
        4
        5
        6
        7
        8
        9
       10
       11
       12
       13
        ⋮
       12
       11
       10
        9
        8
        7
        6
        5
        4
        3
        2
        1
```

Figure 17: Создание массива (1, 2, ..., 20, 21, 20, ..., 2, 1) (1-ый способ))

Выполнения задания для самостоятельной работы - 3.3

```
[164]: append!([i for i in 1:21], [j for j in 20:-1:1])
```

```
[164]: 41-element Vector{Int64}:
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
  ⋮
 12
 11
 10
  9
  8
  7
  6
  5
  4
  3
  2
  1
```

Figure 18: Создание массива (1, 2, ..., 20, 21, 20, ..., 2, 1) (2-ой способ))

Выполнения задания для самостоятельной работы - 3.4

3.4. массив с именем tmp вида (4,6,3); (fig. 19)

```
[91]: #4)
      tmp = rand(4,6,3)
```

```
[91]: 4×6×3 Array{Float64, 3}:
      [:, :, 1] =
       0.22539    0.823737   0.665588  0.867232   0.0812585  0.771429
       0.0298163  0.23293    0.0968    0.740822   0.698985   0.000895025
       0.688696   0.0922611  0.24061   0.994667   0.254193   0.751649
       0.91861    0.642699   0.152821  0.0233591  0.990343   0.909852

      [:, :, 2] =
       0.167417  0.147863  0.633164  0.860159  0.281131   0.335424
       0.870791  0.351263  0.218078  0.890443  0.0787401  0.0345359
       0.902846  0.036589  0.555971  0.844363  0.206415   0.976347
       0.592511  0.676162  0.399539  0.904641  0.250451   0.907981

      [:, :, 3] =
       0.485854  0.819319   0.0303418  0.748488   0.983264  0.0732588
       0.213279  0.395082   0.132885   0.877686   0.328408  0.822766
       0.42471   0.0617085  0.707469   0.225382   0.59201   0.227713
       0.11433   0.473567   0.216158   0.0846277  0.898807  0.907894
```

```
[92]: tmp = rand(4*6*3)
      tmp = reshape(tmp, (4,6,3))
```

```
[92]: 4×6×3 Array{Float64, 3}:
      [:, :, 1] =
       0.980892  0.0201232  0.514157  0.457269   0.456321  0.102833
       0.74343   0.308824   0.192523  0.960027   0.883829  0.873872
       0.707595  0.795022   0.460781  0.148293   0.570975  0.630099
       0.777631  0.123396   0.18402   0.0488513  0.919143  0.291544

      [:, :, 2] =
       0.451821  0.732986  0.406866  0.920406   0.787951   0.506432
       0.956721  0.552998  0.96593   0.0517157  0.0796583  0.816998
       0.904932  0.815316  0.367541  0.973002   0.641644   0.855112
       0.331768  0.600887  0.784642  0.349721   0.898262   0.0618003

      [:, :, 3] =
       0.658466  0.815065  0.787332  0.297999  0.125613  0.522754
       0.543201  0.102713  0.500018  0.019029  0.135879  0.728455
       0.921976  0.100955  0.345157  0.934015  0.499408  0.786769
       0.817066  0.528364  0.859519  0.742694  0.482367  0.777953
```


Figure 19: Создание массива tmp вида (4,6,3) 2-мя способами

Выполнения задания для самостоятельной работы - 3.5

3.5. массив, в котором первый элемент массива tmp повторяется 10 раз; (fig. 20)

```
[93]: #5)
      fill(tmp[1], 10)

[93]: 10-element Vector{Float64}:
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434

[95]: repeat([tmp[1]], 10)

[95]: 10-element Vector{Float64}:
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
```

Figure 20: Создание массива, в котором первый элемент массива tmp повторяется 10 раз, 2-мя способами

Выполнения задания для самостоятельной работы - 3.6

3.6. массив, в котором все элементы массива tmp повторяются 10 раз; (fig. 21-fig. 22)

```
[96]: #6)
      fill(tmp, 10)
```

```
[96]: 10-element Vector{Array{Float64, 3}}:
 [0.9808917867288434 0.020123188447783047 … 0.45632142406202014 0.10283265015
022502; 0.7434301085294002 0.30882397712043275 … 0.8838293508654596 0.8738715
541593599; 0.7075951400097695 0.795022004426991 … 0.5709747571885091 0.630099
0834049478; 0.7776309241296008 0.12339558195359324 … 0.9191428286758081 0.291
5435501093566;;; 0.4518209596910817 0.7329861102593678 … 0.787950988756123 0.
5064317764997949; 0.9567210460434802 0.5529979255726463 … 0.07965832960309915
0.8169975822798834; 0.904931736158913 0.8153164154141312 … 0.6416439865074085
0.8551119546083065; 0.33176764291879257 0.6008867207723797 … 0.89826231629857
29 0.06180025049687332;;; 0.6584657350247354 0.815064943495369 … 0.1256131362
1353853 0.5227537178005716; 0.5432014929540012 0.10271269074213529 … 0.135878
9944805312 0.7284551487825051; 0.921975735677537 0.10095473066123795 … 0.4994
0826113426307 0.7867694671098151; 0.8170661599847463 0.5283639758606172 … 0.4
8236749329329853 0.7779534900213554]
 [0.9808917867288434 0.020123188447783047 … 0.45632142406202014 0.10283265015
022502; 0.7434301085294002 0.30882397712043275 … 0.8838293508654596 0.8738715
541593599; 0.7075951400097695 0.795022004426991 … 0.5709747571885091 0.630099
0834049478; 0.7776309241296008 0.12339558195359324 … 0.9191428286758081 0.291
5435501093566;;; 0.4518209596910817 0.7329861102593678 … 0.787950988756123 0.
5064317764997949; 0.9567210460434802 0.5529979255726463 … 0.07965832960309915
0.8169975822798834; 0.904931736158913 0.8153164154141312 … 0.6416439865074085
0.8551119546083065; 0.33176764291879257 0.6008867207723797 … 0.89826231629857
29 0.06180025049687332;;; 0.6584657350247354 0.815064943495369 … 0.1256131362
1353853 0.5227537178005716; 0.5432014929540012 0.10271269074213529 … 0.135878
9944805312 0.7284551487825051; 0.921975735677537 0.10095473066123795 … 0.4994
0826113426307 0.7867694671098151; 0.8170661599847463 0.5283639758606172 … 0.4
8236749329329853 0.7779534900213554]
 [0.9808917867288434 0.020123188447783047 … 0.45632142406202014 0.10283265015
```

Figure 21: Создание массива, в котором все элементы массива tmp повторяются 10 раз (1-ый способ)

Выполнения задания для самостоятельной работы - 3.6

```
[97]: repeat([tmp], 10)

[97]: 10-element Vector{Array{Float64, 3}}:
 [0.9808917867288434 0.020123188447783047 … 0.45632142406202014 0.10283265015
022502; 0.7434301085294002 0.30882397712043275 … 0.8838293508654596 0.8738715
541593599; 0.7075951400097695 0.795022004426991 … 0.5709747571885091 0.630099
0834049478; 0.7776309241296008 0.12339558195359324 … 0.9191428286758081 0.291
5435501093566;;; 0.4518209596910817 0.7329861102593678 … 0.787950988756123 0.
5064317764997949; 0.9567210460434802 0.5529979255726463 … 0.07965832960309915
0.8169975822798834; 0.904931736158913 0.8153164154141312 … 0.6416439865074085
0.8551119546083065; 0.33176764291879257 0.6008867207723797 … 0.89826231629857
29 0.06180025049687332;;; 0.6584657350247354 0.815064943495369 … 0.1256131362
1353853 0.5227537178005716; 0.5432014929540012 0.10271269074213529 … 0.135878
9944805312 0.7284551487825051; 0.921975735677537 0.10095473066123795 … 0.4994
0826113426307 0.7867694671098151; 0.8170661599847463 0.5283639758606172 … 0.4
8236749329329853 0.7779534900213554]
 [0.9808917867288434 0.020123188447783047 … 0.45632142406202014 0.10283265015
022502; 0.7434301085294002 0.30882397712043275 … 0.8838293508654596 0.8738715
541593599; 0.7075951400097695 0.795022004426991 … 0.5709747571885091 0.630099
0834049478; 0.7776309241296008 0.12339558195359324 … 0.9191428286758081 0.291
5435501093566;;; 0.4518209596910817 0.7329861102593678 … 0.787950988756123 0.
5064317764997949; 0.9567210460434802 0.5529979255726463 … 0.07965832960309915
0.8169975822798834; 0.904931736158913 0.8153164154141312 … 0.6416439865074085
0.8551119546083065; 0.33176764291879257 0.6008867207723797 … 0.89826231629857
29 0.06180025049687332;;; 0.6584657350247354 0.815064943495369 … 0.1256131362
1353853 0.5227537178005716; 0.5432014929540012 0.10271269074213529 … 0.135878
9944805312 0.7284551487825051; 0.921975735677537 0.10095473066123795 … 0.4994
0826113426307 0.7867694671098151; 0.8170661599847463 0.5283639758606172 … 0.4
8236749329329853 0.7779534900213554]
 [0.9808917867288434 0.020123188447783047 … 0.45632142406202014 0.10283265015
022502; 0.7434301085294002 0.30882397712043275 … 0.8838293508654596 0.8738715
```

Figure 22: Создание массива, в котором все элементы массива tmp повторяются 10 раз (2-ой способ)

Выполнения задания для самостоятельной работы - 3.7

3.7. массив, в котором первый элемент массива tmp встречается 11 раз, второй элемент — 10 раз, третий элемент — 10 раз; (fig. 23-fig. 24)

```
[98]: #7)
      append!(fill(tmp[1], 11), append!(fill(tmp[2], 10), fill(tmp[3], 10)))

[98]: 31-element Vector{Float64}:
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.7434301085294002
       0.7434301085294002
       ⋮
       0.7434301085294002
       0.7434301085294002
       0.7075951400097695
       0.7075951400097695
       0.7075951400097695
       0.7075951400097695
       0.7075951400097695
       0.7075951400097695
       0.7075951400097695
       0.7075951400097695
       0.7075951400097695

[100]: append!(repeat([tmp[1]], 11), append!(repeat([tmp[2]], 10),
              repeat([tmp[3]], 10)))

[100]: 31-element Vector{Float64}:
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.9808917867288434
       0.7434301085294002
       0.7434301085294002
       ⋮
```

Figure 23: Создание массива, в котором первый элемент массива tmp встречается 11 раз, второй элемент — 10 раз, третий элемент — 10 раз, 1-2 способами

Выполнения задания для самостоятельной работы - 3.7

```
[101]: append!(repeat([tmp[1], tmp[2], tmp[3]], 10), tmp[1])
```

```
[101]: 31-element Vector{Float64}:
 0.9808917867288434
 0.7434301085294002
 0.7075951400097695
 0.9808917867288434
 0.7434301085294002
 0.7075951400097695
 0.9808917867288434
 0.7434301085294002
 0.7075951400097695
 0.9808917867288434
 0.7434301085294002
 0.7075951400097695
 0.9808917867288434
 ⋮
 0.7434301085294002
 0.7075951400097695
 0.9808917867288434
 0.7434301085294002
 0.7075951400097695
 0.9808917867288434
 0.7434301085294002
 0.7075951400097695
 0.9808917867288434
 0.7434301085294002
 0.7075951400097695
 0.9808917867288434
```

Figure 24: Создание массива, в котором первый элемент массива tmp встречается 11 раз, второй элемент — 10 раз, третий элемент — 10 раз, 3 способ

Выполнения задания для самостоятельной работы - 3.8

3.8. массив, в котором первый элемент массива tmp встречается 10 раз подряд, второй элемент — 20 раз подряд, третий элемент — 30 раз подряд; (fig. 25)

```
[102]: #8)
append!(fill(tmp[1], 10), append!(fill(tmp[2], 20), fill(tmp[3], 30)))

[102]: 60-element Vector{Float64}:
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.7434301085294002
 0.7434301085294002
 0.7434301085294002
 ⋮
 0.7075951400097695
 0.7075951400097695
 0.7075951400097695
 0.7075951400097695
 0.7075951400097695
 0.7075951400097695
 0.7075951400097695
 0.7075951400097695
 0.7075951400097695
 0.7075951400097695
 0.7075951400097695

[103]: append!(repeat([tmp[1]], 10), append!(repeat([tmp[2]], 20),
              repeat([tmp[3]], 30)))

[103]: 60-element Vector{Float64}:
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.9808917867288434
 0.7434301085294002
 0.7434301085294002
 0.7434301085294002
 ⋮
```

Figure 25: Создание массива, в котором первый элемент массива tmp встречается 0 раз подряд, второй элемент — 20 раз подряд, третий элемент —

Выполнения задания для самостоятельной работы - 3.9

3.9. массив из элементов вида $2^{tmp[i]}$, i = 1, 2, 3, где элемент $2^{tmp[3]}$ встречается 4 раза; посчитали в полученном векторе, сколько раз встречается цифра 6; (fig. 26)

```
[105]: #9)
       ar2 = [2^tmp[i] for i in 1:3]
       ar2 = append!(ar2, fill(2^tmp[3], 3))

[105]: 6-element Vector{Float64}:
        1.9736850445967162
        1.6741515232810351
        1.6330796307567377
        1.6330796307567377
        1.6330796307567377
        1.6330796307567377

[106]: ar2 = [2^tmp[i] for i in [1,2,3,3,3,3]]

[106]: 6-element Vector{Float64}:
        1.9736850445967162
        1.6741515232810351
        1.6330796307567377
        1.6330796307567377
        1.6330796307567377
        1.6330796307567377

[109]: n = 0
       for i in (1:length(ar2))
           t = ar2[i]
           k = 1
           if (trunc(Int,t)==0)
               k = 0
           end
           while(!(t%1==0))
               t = t*10
               k = k+1
           end
           t = trunc(Int,t)
           for i in (1:1:k)
               if (t%10==6)
                   n = n+1
               end
               t = div(t, 10)
           end
       end

       n

[109]: 16
```

Figure 26: Создание массива, из элементов вида $2^{tmp[i]}$, i = 1, 2, 3, где элемент $2^{tmp[3]}$ встречается 4 раза; подсчет сколько раз встречается цифра 6 в этом векторе

Выполнения задания для самостоятельной работы - 3.10

3.10. вектор значениий $y = e^x cos(x)$ в точках $x = 3, 3.1, 3.2, ..., 6$, нашли среднее значение y; (fig. 27)

```
[115]: #10)
       y = [exp(i)*cos(i) for i in 3:0.1:6]

[115]: 31-element Vector{Float64}:
        -19.884530844146987
        -22.178753389342127
        -24.490696732801293
        -26.77318244299338
        -28.969237768093574
        -31.011186439374516
        -32.819774760338504
        -34.30336011037369
        -35.35719361853035
        -35.86283371230767
        -35.68773248011913
        -34.68504225166807
        -32.693695428321746
          ⋮
         25.046704998273004
         42.09920106253839
         61.99663027669454
         84.92906736250268
        111.0615860420258
        140.5250750527875
        173.40577640857734
        209.73349424783467
        249.46844055885668
        292.4867067371223
        338.5643778585117
        387.36034029093076

[116]: sum(y)/length(y)

[116]: 53.11374594642971
```

Figure 27: Создание вектора значениий $y = e^x cos(x)$ в точках $x = 3, 3.1, 3.2, ..., 6$; подсчет среднего значения y

Выполнения задания для самостоятельной работы - 3.11

3.11. вектор вида (x^i, y^j), $x = 0.1$, $i = 3, 6, 9, ..., 36$, $y = 0.2$, $j = 1, 4, 7, ..., 34$; (fig. 28)

```
[117]: #11)
       x = 0.1
       y = 0.2

       [[x^i y^(i-2)] for i in 3:3:36]
```

```
[117]: 12-element Vector{Matrix{Float64}}:
        [0.0010000000000000002 0.2]
        [1.0000000000000004e-6 0.0016000000000000003]
        [1.0000000000000005e-9 1.2800000000000006e-5]
        [1.0000000000000008e-12 1.0240000000000006e-7]
        [1.0000000000000009e-15 8.192000000000005e-10]
        [1.0000000000000008e-18 6.553600000000055e-12]
        [1.0000000000000012e-21 5.2428800000000056e-14]
        [1.0000000000000012e-24 4.194304000000005e-16]
        [1.0000000000000015e-27 3.3554432000000044e-18]
        [1.0000000000000017e-30 2.684354560000004e-20]
        [1.0000000000000018e-33 2.147483648000004e-22]
        [1.000000000000002e-36 1.7179869184000035e-24]
```

Figure 28: Создание вектора вида (x^i, y^j), $x = 0.1$, $i = 3, 6, 9, ..., 36$, $y = 0.2$, $j = 1, 4, 7, ..., 34$

Выполнения задания для самостоятельной работы - 3.12

3.12. вектор с элементами $\frac{2^i}{i}, i = 1, 2, ..., M, M = 25$; (fig. 29)

```
[118]: #12)
       M = 25
       [2^i/i for i in 1:1:M]

[118]: 25-element Vector{Float64}:
             2.0
             2.0
             2.6666666666666665
             4.0
             6.4
            10.666666666666666
            18.285714285714285
            32.0
            56.888888888888886
           102.4
           186.1818181818182
           341.3333333333333
           630.1538461538462
          1170.2857142857142
          2184.5333333333333
          4096.0
          7710.117647058823
         14563.555555555555
         27594.105263157893
         52428.8
         99864.38095238095
        190650.18181818182
        364722.0869565217
        699050.6666666666
             1.34217728e6
```

Figure 29: Создание вектора вида $c элементами $\frac{2^i}{i}, ii = 1, 2, ..., 25$

Выполнения задания для самостоятельной работы - 3.13

3.13. вектор вида (”fn1”,”fn2”,...,”fnN”) , N=30; (fig. 30)

```
[119]: #13)
       N = 30
       [string("fn", string(i)) for i in 1:N]
```

```
[119]: 30-element Vector{String}:
        "fn1"
        "fn2"
        "fn3"
        "fn4"
        "fn5"
        "fn6"
        "fn7"
        "fn8"
        "fn9"
        "fn10"
        "fn11"
        "fn12"
        "fn13"
        ⋮
        "fn19"
        "fn20"
        "fn21"
        "fn22"
        "fn23"
        "fn24"
        "fn25"
        "fn26"
        "fn27"
        "fn28"
        "fn29"
        "fn30"
```

Figure 30: Создание вектора вида (”fn1”,”fn2”,...,”fn30”)

Выполнения задания для самостоятельной работы - 3.14

3.14. векторы $x = (x_1, x_2, ..., x_n)$ и $y = (y_1, y_2, ..., y_n)$ целочисленного типа длины n = 250, как случайные выборки из совокупности 0, 1, ... , 999; на его основе: (fig. 31-fig. 43)

– сформировали вектор $(y_2 - x_1, ..., y_n - x_{n-1})$;

```
[120]: #14)
       n = 250
       x = rand(0:999, n)
       y = rand(0:999, n)
       ...

[121]: [y[i+1]-x[i] for i in 1:n-1]

[121]: 249-element Vector{Int64}:
         554
        -618
        -421
         238
          24
         168
         215
         172
         259
        -191
        -157
        -469
        -833
           ⋮
         -60
        -442
        -106
         165
        -496
        -547
        -488
         770
        -437
         683
         109
         796
```

Figure 31: Создание векторов x, y, $((y_2 - x_1, ..., y_n - x_{n-1})$

Выполнения задания для самостоятельной работы - 3.14

- сформировали вектор
$(x_1 + 2x_2 - x_3, x_2 + 2x_3 - x_4, ..., x_{n-2} + 2x_{n-1} - x_n)$;

```
[122]: [x[n-2]+2*x[n-1]-x[i] for i in 3:n]

[122]: 248-element Vector{Int64}:
 -711
 -532
 -316
  113
 -589
 -513
 -428
 -653
 -273
 -845
 -812
 -439
   18
    ⋮
 -627
 -107
  -17
 -381
 -617
 -752
   71
 -432
   18
   64
  103
 -783
```

Figure 32: Создание вектора
$(x_1 + 2x_2 - x_3, x_2 + 2x_3 - x_4, ..., x_{n-2} + 2x_{n-1} - x_n)$

Выполнения задания для самостоятельной работы - 3.14

- сформировали вектор $\left(\frac{sin(y_1)}{cos(x_2)}, \frac{sin(y_2)}{cos(x_2)}, \ldots, \frac{sin(y_{n-1})}{cos(x_n)}\right)$;

```
[123]: [sin(y[i-1])/cos(x[i]) for i in 2:n]

[123]: 249-element Vector{Float64}:
 -0.6601300937044344
  1.0396471609189781
 -1.287957059948607
 -4.288536785917236
 -0.21952208609748033
 -4.248217489539866
  1.481595686747271
 -0.4159697163707091
  0.051645077729180824
 -0.9691530321343063
  0.0987836690021917
  1.7011892063534413
 -1.4393997926578683
  ⋮
 -1.2930981885868857
  0.5465846025946713
 -1.1924863971287591
 -1.1105433496420303
 -0.7409790344605799
  1.90451099365609
 -1.824382586954184
 -0.30869330656435034
  1.3750813882088726
  3.009830233868485
  1.0716190537438743
 -1.0101771371185286
```

Figure 33: Создание вектора $\left(\frac{sin(y_1)}{cos(x_2)}, \frac{sin(y_2)}{cos(x_2)}, \ldots, \frac{sin(y_{n-1})}{cos(x_n)}\right)$

Выполнения задания для самостоятельной работы - 3.14

- вычислили $\sum_{i=1}^{n-1} \frac{e^{-x_{i+1}}}{x_i+10}$;

```
[125]: s = 0
       for i in 1:n-1
           s = s + exp(-x[i+1])/(x[i]+10)
       end
       s

[125]: 0.0012053137137940 68
```

Figure 34: Вычисление $\sum_{i=1}^{n-1} \frac{e^{-x_{i+1}}}{x_i+10}$

Выполнения задания для самостоятельной работы - 3.14

- выбрали элементы вектора y, значения которых больше 600; определили индексы этих элементов;

```
[130]: y600 = [y[i] for i in 1:n if (y[i]>600)]

[130]: 99-element Vector{Int64}:
        651
        905
        939
        820
        822
        900
        874
        816
        979
        747
        658
        951
        848
          ⋮
        899
        694
        926
        759
        900
        948
        954
        702
        674
        834
        800
        828

[131]: ind = findall(y .> 600)

[131]: 99-element Vector{Int64}:
          2
          5
          8
          9
         10
         16
         17
         20
         23
         26
         27
         28
         29
          ⋮
```

Figure 35: Создание вектора из вектора y со значениями больше 600 и

Выполнения задания для самостоятельной работы - 3.14

- определили значения вектора x, соответствующие значениям вектора y, значения которых больше 600 (под соответствием понимается расположение на аналогичных индексных позициях);

```
[132]: [x[i] for i in ind]

[132]: 99-element Vector{Int64}:
 823
 451
 648
 563
 788
 179
 945
 354
 883
 482
 609
 937
 717
   ⋮
 913
 553
  87
 836
 696
 847
 447
 300
 492
 567
  71
 918
```

Figure 36: Создание вектора из вектора x, по соответсвию с элементами вектора y больше 600

Выполнения задания для самостоятельной работы - 3.14

- сформировали вектор $(|x_1 - \bar{x}|^{\frac{1}{2}}, |x_2 - \bar{x}|^{\frac{1}{2}}, ..., |x_n - \bar{x}|^{\frac{1}{2}})$, где $\bar{x}$ обозначает среднее значение вектора x

```
[133]: avg_x = sum(x)/length(x)
       [sqrt(abs(x[i]-avg_x)) for i=1:n]

[133]: 250-element Vector{Float64}:
        20.355834544424848
        17.65332829808589
        18.2931681236466
        12.475576139000555
         7.7691698398220135
        22.121482771279144
        14.582180906846547
        11.689311356961966
         7.186097689288672
        16.632498309033434
        10.166612021711067
        21.648094604375693
        20.87199080107118
         ⋮
        15.831613941730641
        16.412190591142913
        18.956792977716457
         2.1540659228537984
        15.512575543732252
        19.381434415439948
        21.15088650624366
         7.459222479588606
        19.85849943978648
        20.984756372185977
        21.894291493446413
        20.16531675922796
```

Figure 37: Создание вектора из вектора $(|x_1 - \bar{x}|^{\frac{1}{2}}, |x_2 - \bar{x}|^{\frac{1}{2}}, ..., |x_n - \bar{x}|^{\frac{1}{2}})$

Выполнения задания для самостоятельной работы - 3.14

- определили, сколько элементов вектора y отстоят от максимального значения не более, чем на 200;

```
[135]: y_avg = sum(y)/length(y)
       length([y[i] for i=1:n if (abs(y[i]-y_avg)<=200 && y[i]<y_avg)])

[135]: 51
```

Figure 38: Определение, сколько элементов вектора y отстоят от максимального значения не более, чем на 200

Выполнения задания для самостоятельной работы - 3.14

- определили, сколько чётных и нечётных элементов вектора x;

```
[137]: even_x = length([x[i] for i=1:n if (x[i]%2==0)])
       uneven_x = length([x[i] for i=1:n if (x[i]%2!=0)])
       even_x, uneven_x

[137]: (130, 120)
```

Figure 39: Определение, сколько чётных и нечётных элементов вектора x

Выполнения задания для самостоятельной работы - 3.14

- определили, сколько элементов вектора x кратны 7;

```
[138]: length([x[i] for i=1:n if (x[i]%7==0)])

[138]: 37
```

Figure 40: Определение, сколько сколько элементов вектора x кратны 7

Выполнения задания для самостоятельной работы - 3.14

- отсортировали элементы вектора x в порядке возрастания элементов вектора y;

```
[152]: y_sorted = sort!(y)
       y_s_ind = [ indexin(y_sorted[i],y)[1] for i=1:n]
       x_s = [x[i] for i in y_s_ind]

[152]: 250-element Vector{Int64}:
        97
       823
       846
       667
       451
        22
       724
       724
       563
       788
       408
       408
       947
         ⋮
       762
       242
       152
       516
       752
       887
        64
       567
       567
        71
        32
       918
```

Figure 41: Создание вектора на основе вектора x, отсортированного в порядке возрастания элементов вектора y

Выполнения задания для самостоятельной работы - 3.14

- вывели элементы вектора x, которые входят в десятку наибольших;

```
[153]: sort!(x, rev=true)[1:10]

[153]: 10-element Vector{Int64}:
        998
        992
        987
        984
        981
        980
        980
        973
        961
        947
```

Figure 42: Вывод элементов вектора x, которые входят в десятку наибольших

Выполнения задания для самостоятельной работы - 3.14

- сформировали вектор, содержащий только уникальные (неповторяющиеся) элементы вектора x.

```
[154]: unique(x)

[154]: 226-element Vector{Int64}:
 998
 992
 987
 984
 981
 980
 973
 961
 947
 946
 945
 937
 926
   ⋮
  49
  48
  44
  40
  32
  31
  24
  22
  18
  11
  10
   0
```

Figure 43: Создание вектора, содержащего только уникальные (неповторяющиеся) элементы вектора x

4. Создали массив squares, в котором хранятся квадраты всех целых чисел от 1 до 100. (fig. 44)

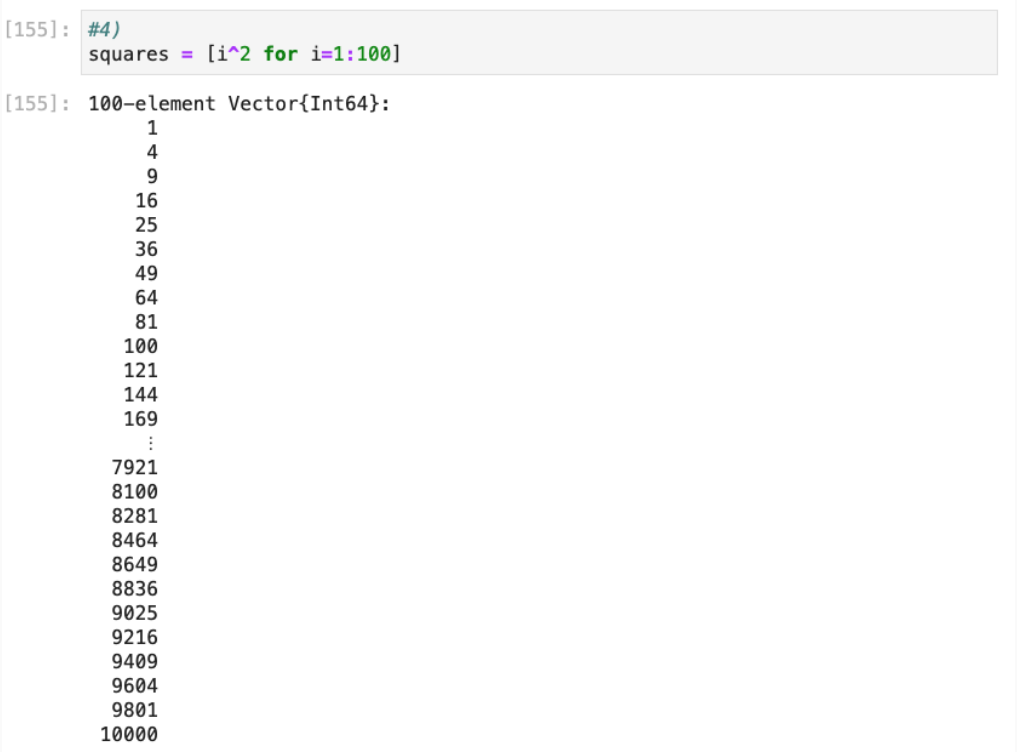
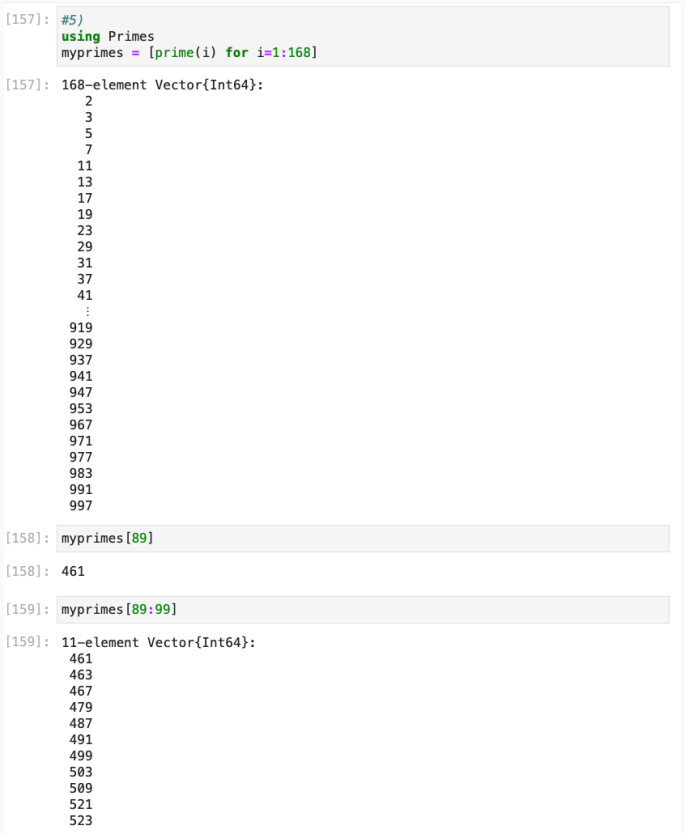


Figure 44: Создание массива squares

Выполнения задания для самостоятельной работы - 3.5

5. Сгенерировали массив myprimes, в котором хранятся первые 168 простых чисел. Определили 89-е наименьшее простое число. Получили срез массива с 89-го до 99-го элемента включительно, содержащий наименьшие простые числа. (fig. 45)



```
[157]: #5)
       using Primes
       myprimes = [prime(i) for i=1:168]

[157]: 168-element Vector{Int64}:
         2
         3
         5
         7
        11
        13
        17
        19
        23
        29
        31
        37
        41
         ⋮
       919
       929
       937
       941
       947
       953
       967
       971
       977
       983
       991
       997

[158]: myprimes[89]

[158]: 461

[159]: myprimes[89:99]

[159]: 11-element Vector{Int64}:
       461
       463
       467
       479
       487
       491
       499
       503
       509
       521
       523
```


Figure 45: Создание и работа с массивом myprimes

6. Вычислили следующие выражения: (fig. 46-fig. 48)

6.1. $\sum_{i=10}^{100}(i^3+4i^2)$

```
[160]: #6.

       #1)
       s = 0
       for i=10:100
           s = s + i^3+4*i^2
       end
       s
```

```
[160]: 26852735
```

Figure 46: Вычисление выражения 6.1

6.2. $\sum_{i=1}^{M}(\frac{2^i}{i} + \frac{3^i}{i^2}), M = 25$

```
[161]: #2)
       M = 25
       s = 0
       for i=1:M
           s = s + 2^i/i + 3^i/i^2
       end
       s
```

```
[161]: 2.1291704368143802e9
```

Figure 47: Вычисление выражения 6.2

6.3. $1 + \frac{2}{3} + (\frac{2}{3}\frac{4}{5} + ... + \frac{2}{3}\frac{4}{5}...\frac{38}{39}))$

```
[162]: #3)
       s = 1
       t = 1
       for i=2:2:38
           t = t * i / (i+1)
           s = s + t
       end
       s
```

```
[162]: 6.97634613789762
```

Figure 48: Вычисление выражения 6.3

Результаты выполнения лабораторной работы

Результаты выполнения лабораторной работы

- были изучены структуры данных в Julia и операции над ними
- программа на языке Julia, работающая со структурами данных